UNITED STATES PATENT APPLICATION

METHOD AND APPARATUS FOR INDIRECTLY ADDRESSED VECTOR LOAD-ADD-STORE ACROSS MULTI-PROCESSORS

INVENTOR:

James R. Kohn   Citizenship: **USA**   Residence: **Inver Grove Heights, MN.**
Post Office Address: **10165 Adam Avenue, Inver Grove Heights, MN. 55077**

Schwegman, Lundberg, Woessner, & Kluth, P.A.
1600 TCF Tower
121 South Eighth Street
Minneapolis, Minnesota 55402
Attorney Docket 01376.731us1

1

1   **METHOD AND APPARATUS FOR INDIRECTLY ADDRESSED VECTOR**

2   **LOAD-ADD-STORE ACROSS MULTI-PROCESSORS**

3

4

5

6   <u>**Related Applications**</u>

7   This application is related to U.S. Patent Application No. _____,

8   entitled "Multistream Processing System and Method", filed on even date herewith;

9   to U.S. Patent Application No. _____, entitled "System and Method for

10   Synchronizing Memory Transfers", Serial No. _____, filed on even date

11   herewith; to U.S. Patent Application No. _____, entitled "Decoupled

12   Store Address and Data in a Multiprocessor System", filed on even date herewith; to

13   U.S. Patent Application No. _____, entitled "Decoupled Vector

14   Architecture", filed on even date herewith; to U.S. Patent Application No.

15   _____, entitled "Latency Tolerant Distributed Shared Memory

16   Multiprocessor Computer", filed on even date herewith; to U.S. Patent Application

17   No. _____, entitled "Relaxed Memory Consistency Model", filed on even

18   date herewith; to U.S. Patent Application No. _____, entitled "Remote

19   Translation Mechanism for a Multinode System", filed on even date herewith; and to

20   U.S. Patent Application No. _____, entitled "Method and Apparatus for

21   Local Synchronizations in a Vector Processor System", filed on even date herewith,

22   each of which is incorporated herein by reference.

23

24   <u>**Field of the Invention**</u>

25   This invention relates to the field of vector computers, and more specifically

26   to a method and apparatus to correctly computer a vector-load, vector-operate (such

27   as a vector add), and vector-store sequence, particularly when elements of the vector

28   may be redundantly presented as in the case of indirectly addressed vector operations

29   from and to memory.

1

Background of the Invention

3   Indirectly addressed operands are frequently used in computer programs. For

4   example, one typical situation provides a load instruction that specifies a register

5   having an address of an operand in memory (rather than the address being partially

6   or completely specified directly by the instruction), and another register that is the

7   destination of the operand being fetched or loaded. A store instruction using indirect

8   addressing would similarly specify a register that holds the address in memory of the

9   destination, and another register that is the source of the operand being stored.

10   Vector computers provide a fast and compact way of programming for codes

11   that are amenable to vectorizing to improve speed and programming efficiency.

12   What is needed is a fast, repeatable, and accurate way of performing various

13   indirectly addressed operations in a vector computer.

14

Summary of the Invention

16   The present invention provides a method and apparatus to correctly compute

17   a vector-gather, vector-operate (e.g., vector add), and vector-scatter sequence,

18   particularly when elements of the vector may be redundantly presented, as with

19   indirectly addressed vector operations. For an add operation, one vector register is

20   loaded with the "add-in" values, and another vector register is loaded with address

21   values of "add to" elements to be gathered from memory into a third vector register.

22   If the vector of address values has a plurality of elements that point to the same

23   memory address, the algorithm should add all the "add in" values from elements

24   corresponding to the elements having the duplicated addresses. An indirectly

25   addressed load performs the "gather" operation to load the "add to" values. A vector

26   add operation then adds corresponding elements from the "add in" vector to the "add

27   to" vector. An indirectly addressed store then performs the "scatter" operation to

28   store the results.

29

30

1   **Brief Description of the Drawings**

2   FIG. 1A shows a block diagram of one embodiment of the present invention having a

3   vector processing system 100.

4   FIG. 1B shows a block diagram of further aspects of vector processing system 100.

5   FIG. 1C shows a block diagram of an MSP 102 of some embodiments of the present
6   invention.

7   FIG. 1D shows a block diagram of a node 106 of some embodiments of the present
8   invention.

9   FIG. 1E   shows a block diagram of a system 108 of some embodiments of the
10   present invention.

11

12

13   **Description of Preferred Embodiments**

14   In the following detailed description of the preferred embodiments, reference

15   is made to the accompanying drawings that form a part hereof, and in which are

16   shown by way of illustration specific embodiments in which the invention may be

17   practiced. It is understood that other embodiments may be utilized and structural

18   changes may be made without departing from the scope of the present invention.

19   The leading digit(s) of reference numbers appearing in the Figures generally

20   corresponds to the Figure number in which that component is first introduced, such

21   that the same reference number is used throughout to refer to an identical component

22   which appears in multiple Figures. The same reference number or label may refer to

23   signals and connections, and the actual meaning will be clear from its use in the

24   context of the description.

25   In some embodiments, there is a software application invention that requires

26   that particular sequence of operations to maintain order. The algorithm itself has

27   been vectorized. There is a requirement that within a vector of more than one

28   element, since there may be collisions in the memory system where its referencing

29   the same memory location multiple times in the vector. There needs to be a

30   guarantee that updates to that memory location are done in order. In some

31   embodiments, each memory location is an 8-byte memory location. In some

32   embodiments, there is a vector instruction that can operate on multiple 8 byte

1   quantities in one instruction, by using one vector register to hold addresses of the

2   elements to be loaded into another vector register, that may not be contiguous in

3   memory. In fact in this particular case they are often not contiguous.

4   Memory location can be referenced with a single instruction. But there may

5   be multiple occurrences of any given memory address in that single instruction and

6   now we're trying to do like a add of a value to that memory location. And when an

7   addition operation occurs multiple times, there is the possibility of losing one of the

8   adds or getting the adds out-of-order. This has been a known vectorization problem.

9   There are generally three instructions of interest. There is a load operation

10   which loads the existing memory contents or a number of elements greater than one,

11   into a vector register using indirect addressing. Then there's an add operation that

12   wants to add a set of values to those elements that are loaded, such as V1 is assigned

13   V2 plus V1. Then we want to store the result back out into the same memory

14   location. And if the memory locations are all disjoint, this can occur at full speed in

15   the vector hardware of Figure 1D described below. The problem occurs, for which

16   this special algorithm is needed, is when there are overlapping or multiple

17   occurrences of the same memory location in the vector register used for addressing.

18   The original values are loaded into v1. Now we add v2 to v1. In conventional

19   methods, the first element that has multiple instances of the address is correct, but

20   the additions after that are or can be incorrect because they lose the previous

21   additions. So when we store the final result of the add back out to memory, we get

22   an incorrect answer in memory. Thus, we need a method to recognize where the

23   conflicting memory locations are, and we have such an algorithm for older systems,

24   and part of the application is probably going to have to describe that old algorithm.

25   And then for the X1 that old algorithm did not work very well, the present invention

26   provides a new way of detecting those collisions.

27   In one conventional algorithm, after you did that load from memory, you

28   would use the same memory location to store back a known pattern and then you

29   would load back that pattern and do a comparison against the original pattern and if

1   they matched then there were no collisions. But if they didn't match that means that

2   one or more locations, or that a location had more than one store into it.

3   The other vector register specifies an index to those locations. And it's those

4   indexes that may be repeated. That index is used both for the load as well as the

5   store back later.

6   In the old way what you'd do is you'd have a pattern of say 1,2,3,4,5,6,7 in

7   the elements and if you didn't get back, if you got 1,2,2 or 1,6,6. You would see

8   where there was a collision and which elements were colliding. Then you unwrap

9   the vector and do it as individual instructions. Effectively that's the conventional

10 algorithm. The intent is to have a fast way of detecting that we do have a collision.

11 The new algorithm, instead of using the original array that we loaded, storing this

12 1,2,3,4,5 etc., creates a temporary scratch array and uses that instead.

13   In fact, one can delay the load of the elements to be added, since the

14 calculations to determine duplicates only needs the scratch area and the addressing

15 vector register. The algorithm selects a certain number of bits out of the index vector

16 elements, like say 12 bits, it doesn't' really matter how many bits, and use that

17 reduced index of the index into the temporary. Occasionally you get some false

18 positives. The new algorithm addresses how to deal with the false positives. And

19 does it in such a way that performance is improved on the X1 with this new

20 technique.

21   The new algorithm goes on, instead of doing an add like the old algorithm

22 did, it does an add back into the add-in values having duplicated indexes to compress

23 that vector.

24   Figure 1A shows a block diagram of one embodiment of the present

25 invention having a vector-processing system 100. Figure 1B shows a block diagram

26 of further aspects of vector processing system 100.

27   In some embodiments, as shown in Figures 1A and 1B, a first vector register

28 110 having E elements is loaded with the "add-in" values A0, A1, ... A(E-1) into

29 element addresses 0, 1, ... (E-1) of register 110 (i.e., each element of the first vector

30 register 110 is a different value to be added to a corresponding element fetched from

1 memory), and a second vector register 112 is loaded with address values @0, @1, ...

2 @(E-1)(i.e., each element of the second vector register 112 is a different signed

3 offset value to be added to a base address pointer to obtain the virtual address of the

4 corresponding element fetched from memory), of "add to" elements to be gathered

5 from memory 150 (e.g., from a table).

6   Occasionally, a plurality of such addresses will be equal, thus specifying to

7 fetch the same "add to" element to a plurality of locations in the add-to vector

8 register 110. For example, elements 2, 15, and 47 (these are the element addresses of

9 elements in the vector) of the second register 112 might all have the same offset, say

10 60033, and the base register could have the pointer to, say address 500000. Thus, the

11 addresses of elements 2, 15, and 47 would each point to memory address 560033.

12 The elements 2, 15, and 47 of the "add to" vector 110 would all be loaded with the

13 value from memory address 5033.

14   In some embodiments, the desired algorithm would want the same behavior

15 and the same result whether the gather-add-scatter operations were performed one

16 element at a time, 16 elements at a time, 64 elements at a time, or any other number

17 of elements at a time, and regardless of the alignment of the gathered elements

18 relative to the start of any vector operation. Thus, in this example, the value starting

19 in memory address 560033 would be loaded (in a vector "gather" operation), be

20 added to the "add in" values from elements 2, 15, and 47 of the first vector register

21 110, and this combined result would be stored back to memory location 560033 (in a

22 "scatter" operation). In some embodiments, this provides the same result as if the

23 value starting in memory address 560033 would be loaded (in a serial "gather"

24 operation), be added to the "add in" value from element 2 of the first vector register

25 110, and stored back to memory location 560033, then this value from memory

26 address 560033 would be again loaded, be added to the "add in" value from element

27 15 of the first vector register 110, and stored back to memory location 560033, and

28 then this value from memory address 560033 would be again loaded, be added to the

29 "add in" value from element 47 of the first vector register 110, and stored back to

30 memory location 560033.

1   Since the identities of the elements in the second vector register 112 having

2   the same addresses are unknown, the present invention provides a way to determine

3   those elements. In some embodiments, a first sequence of identification values is

4   stored to a series of addressed locations within a constrained area of memory 161.

5   The address of each location used to store the sequence of values in the constrained

6   area 161 is based at least in part on a corresponding one of the addressing values.

7   For example, the constrained area could have 2^N locations (e.g., in some

8   embodiments, $2^N = 2^{12} = 4096$ locations), and N bits (e.g., N=12 bits) of the address

9   value are used as an offset into the constrained area. Continuing with the above

10   example, the address offset 60033 could have any 12 bits extracted. Assume, for

11   example, the low 12 bits are used, which would extract "033" from the 60033 value,

12   assuming hexadecimal number notation. If the constrained area 161 had a base

13   address of 7000, then the location 7033 would be the destination of the identification

14   values for elements 2, 15, and 47, and since they are written in element order,

15   location 7033 would end up with the value stored for element 47.

16   The method then reads back 116 from the sequence of addressed locations

17   values resulting from the storing of the first sequence to obtain a second sequence of

18   values, comparing 118 the first sequence of values to the second sequence of values

19   to generate a bit vector representing compares and miscompares, compressing 120

20   the second vector of operand values using the bit vector, using the first vector of

21   addressing values as masked by the bit vector. I.e., for an add operation, where the

22   redundantly addressed locations point to a single memory value B (m) (i.e., at

23   location 560033), each of the corresponding A elements 2, 15, and 47 are added to

24   that Bm value and the result is stored to location 56033). The method further

25   includes loading 124 a third vector register with elements from memory, performing

26   126 an arithmetic-logical operation using values from the third vector register and

27   the compressed second vector of operand values to generate a result vector, and

28   using the first vector of addressing values as masked by the bit vector, storing 128

29   the result vector to memory.

1   One exemplary program that codes one embodiment of the invention is listed

2 below.

3

4   Figure 1C shows a block diagram of a multistreaming processor (MSP) 102

5 that is usable by the above method, for some embodiments of the present invention.

6 MSP 102 includes a plurality of P chips or P circuits 100 (each representing one

7 single-streaming processor having a plurality of vector pipelines and a scalar

8 pipeline), each P chip/circuit 100 connected to a plurality of E chips or E circuits 101

9 (each representing an external cache, synchronization, and memory-interface

10 function). In some embodiments, every P chip/circuit 100 is connected to every E

11 chip/circuit 101. In some embodiments, four P Chips 100 and four E Chips 101 form

12 one MSP 102. Although the P Chip 100 and the E Chips 101 are sometimes

13 described herein as "chips" as representing one embodiment, in other embodiments,

14 they are implemented with a plurality of chips each, or with a single chip containing

15 a plurality of P circuits 100 and/or E circuits 101.

16   In some embodiments, each scalar processing unit 12 delivers a peak of 0.4

17 GFLOPS and 0.8 GIPS at the target frequency of 400 MHz. Each processor 100

18 contains two vector pipes, running at 800 MHz, providing 3.2 GFLOPS for 64-bit

19 operations and 6.4 GFLOPS for 32-bit operations. The MSP 102 thus provides a

20 total of 3.2 GIPS and 12.8/25.6 GFLOPS. Each processor 100 contains a small

21 Dcache used for scalar references only. A two-MB Ecache 24 is shared by all the

22 processors 100 in MSP 102 and used for both scalar and vector data. In one

23 embodiment, each processor 100 and e-circuit 101 of cache 24 are packaged as

24 separate chips (termed the "P" chip and "E" chips, respectively).

25   In some embodiments, signaling between processor 100 and cache 24 runs at

26 400 Mb/s on processor-to-cache connection 32. Each processor-to-cache connection

27 32 shown in Figure 1C uses an incoming 64-bit path for load data and an outgoing

28 64-bit path for requests and store data. Loads, in some embodiments, can achieve a

29 maximum transfer rate of fifty-one GB/s from cache 24. Stores, in some

1   embodiments, can achieve up to forty-one GB/s for stride-one and twenty-five GB/s

2   for non-unit stride stores.

3   In some embodiments, global memory 26 is distributed to each MSP 102 as

4   local memory 105. Each E Chip 101 has four ports 34 to M chip 104 (and through

5   M chip 104 to local memory 105 and to network 107). In some embodiments, ports

6   34 are sixteen data bits in each direction. MSP 102 has a total of 25.6 GB/s load

7   bandwidth and 12.8-20.5 GB/s store bandwidth (depending upon stride) to local

8   memory.

9   Figure 1D shows a block diagram of a node 106 of some embodiments of the

10   present invention. In some embodiments, a node 106 is packaged on a single

11   printed-circuit board. Node 106 includes a plurality of MSPs 102 each connected to

12   a plurality of M chips 104, each M-chip 104 controlling one or more sections of

13   memory 105. In some embodiments, each M chip 104 is connected to memory 105

14   using a plurality of channels (e.g., eight), each channel having a plurality of direct

15   RAMBUS DRAM chips (e.g., four). In some embodiments, each node also includes

16   a plurality of I/O channels 103 used to connect to a local-area network (e.g., one or

17   more gigabit ethernet connections) and/or storage (e.g., disk storage or a storage-area

18   network). Each node 106 also includes one or more network connections that

19   interconnect the memories of a plurality of nodes, in some embodiments.

20   In some embodiments, each node 106 includes four MSPs 102 and sixteen M

21   chips 104. M chips 104 contain memory controllers, network interfaces and cache

22   coherence directories with their associated protocol engines. In one such

23   embodiment, memory 26 is distributed round-robin by 32-byte cache lines across the

24   sixteen M chips 104 at each node 106. Thus, the M chip for a particular address is

25   selected by bits 8..5 of the physical address.

26   Each E Chip 101 is responsible for one fourth of the physical address space,

27   determined by bits 5 and 6 of the physical address. A reference to a particular line of

28   memory is sent to the associated E Chip 101 where the Ecache is consulted, and

29   either the line is found in the Ecache or the request is sent on to an M chip. Bits 7

1   and 8 of the physical address select one of four M chips connected to each E Chip

2   101.

3   Each M chip 104 resides in one of sixteen independent slices of the machine,

4   and the interconnection network 107 provides connectivity only between

5   corresponding M chips on different nodes (thus there are sixteen parallel,

6   independent networks). All activity (cache, memory, network) relating to a line of

7   memory stays within the corresponding system slice.

8   Each M chip 104 contains two network ports 44, each 1.6 GB/s peak per

9   direction. This provides a total peak network bandwidth of 51.2 GB/s in and 51.2

10 GB/s out. Single transfers to/from any single remote destination will use only half

11 this bandwidth, as only one of two ports 44 per M chip 104 will be used. Also,

12 contention from the other processors 100 on node 106 must be considered. Lastly,

13 all inter-node data is packetized, resulting in a smaller ratio of sustained to peak than

14 in the local memory subsystem. Protocol overheads vary from 33% (one way, stride-

15 1 reads) to 83% (symmetric, non-unit-stride reads or writes).

16   Each node 106 also contains two I/O controller chips 103 ("I" chips) that

17 provide connectivity between the outside world and network 107 and memory 26. In

18 some embodiments, each "I" chip 103 provides two XIO (a.k.a. Crosstalk) I/O

19 channels 49, with a peak speed bandwidth of 1.2 GB/s full duplex each. The I chips

20 are connected to each other and to the sixteen M chips 104 with enough bandwidth to

21 match the four XIO channels.

22   This partitioning provides low latency and high bandwidth to local memory

23 105. With a local memory size of up to sixteen GB (sixty-four GB, once 1 Gbit

24 chips become available), most single-processor and autotasked codes should run

25 locally, and most references in distributed-memory codes will be satisfied locally as

26 well. Latency to remote memory will depend upon the distance to the remote node,

27 and the level of contention in network 107.

28   In some embodiments, a limited operating system executes on each node,

29 with a Unicos/mk-like layer across nodes 106. The limited OS will provide basic

30 kernel services and management of two direct-attached I/O devices (a disk array and

1   network interface). All other I/O connectivity is provided by a separate host system.

2   In one such embodiment, the host system also provides the user environment (shell,

3   cross compilers, utility programs, etc.), and can be used to run scalar compute

4   applications.

5   Figure 1E shows a block diagram of a system 108 of some embodiments of

6   the present invention. System 108 includes a plurality of nodes 106 each connected

7   to a common network 107. In some embodiments, network 107 is also connected to

8   one or more other networks 109.

9   One aspect of the invention provides a computerized method that includes

10   providing a first vector 110 of addressing values, providing a second vector 112 of

11   operand values, storing 114 a first sequence of values to a sequence of addressed

12   locations within a constrained area of memory, wherein each location's address is

13   based at least in part on a corresponding one of the addressing values, reading back

14   116 from the sequence of addressed locations values resulting from the storing of the

15   first sequence to obtain a second sequence of values, comparing 118 the first

16   sequence of values to the second sequence of values to generate a bit vector

17   representing compares and miscompares, compressing 120 the second vector of

18   operand values using the bit vector, using the first vector of addressing values as

19   masked by the bit vector, loading 124 a third vector register with elements from

20   memory, performing 126 an arithmetic-logical operation using values from the third

21   vector register and the compressed second vector of operand values to generate a

22   result vector, and using the first vector of addressing values as masked by the bit

23   vector, storing 128 the result vector to memory.

24   In some embodiments, addresses of the elements in memory are calculated by

25   adding each respective addressing value to a base address of an object in memory.

26   In some embodiments, the arithmetic-logical operation is an addition

27   operation that produces at least one element of the result vector as a summation of an

28   element of the loaded third vector register and a plurality of respective elements of

29   the original second vector of operand values corresponding to elements of the first

30   vector of addressing values that had identical values.

1   In some embodiments, address values for the sequence of addressed locations

2   within the constrained area of memory are each calculated using a truncated portion

3   of each respective addressing value of the first vector of addressing values. In some

4   embodiments, data values of the first sequence of values are each formed by

5   concatenating a portion of each respective addressing value of the first vector of

6   addressing values to a respective one of a sequence of numbers.

7   In some embodiments, the constrained area of memory includes 2^N locations,

8   wherein address values for the sequence of addressed locations within the

9   constrained area of memory are each calculated by adding a base address to an N-bit

10   portion of each respective addressing value of the first vector of addressing values,

11   and wherein data values of the first sequence of values are each formed by

12   concatenating a portion of each respective addressing value of the first vector of

13   addressing values to a respective one of a consecutive sequence of integer numbers.

14   In some embodiments, for the loading of the third vector register with

15   elements from memory, elements are loaded from locations specified by addressing

16   values corresponding to bits of the bit vector that indicated a compare and no

17   elements are loaded from locations specified by addressing values corresponding to

18   bits of the bit vector that indicated a miscompare.

19   In some embodiments, the operations recited therein are executed in the order

20   recited therein.

21   Some embodiments, further include performing 124 a first synchronization

22   operation that ensures that the comparing the first sequence of values to the second

23   sequence of values to generate the bit vector representing compares and

24   miscompares effectively completes before the loading of the third vector register

25   with elements from memory, and performing 130 a second synchronization operation

26   that ensures that the storing the result vector to memory completes before subsequent

27   passes through a loop.

28   Another aspect of the invention provides a computer-readable medium

29   having instructions stored thereon for causing a suitably programmed information-

30   processing system to execute a method that includes providing 110 a first vector of

13

1   addressing values, providing 112 a second vector of operand values, storing 114 a

2   first sequence of values to a sequence of addressed locations within a constrained

3   area of memory, wherein each location's address is based at least in part on a

4   corresponding one of the addressing values, reading back 116 from the sequence of

5   addressed locations values resulting from the storing of the first sequence to obtain a

6   second sequence of values, comparing 118 the first sequence of values to the second

7   sequence of values to generate a bit vector representing compares and miscompares,

8   compressing 120 the second vector of operand values using the bit vector, using the

9   first vector of addressing values as masked by the bit vector, loading 124 a third

10   vector register with elements from memory, performing 126 an arithmetic-logical

11   operation using values from the third vector register and the compressed second

12   vector of operand values to generate a result vector, and using the first vector of

13   addressing values as masked by the bit vector, storing 128 the result vector to

14   memory.

15   Yet another aspect of the invention provides a computerized method that

16   includes loading 210 a first vector register with addressing values, loading 212 a

17   second vector register with operand values, storing 214 a first sequence of values to a

18   sequence of addressed locations within a constrained area of memory, wherein each

19   one of these location's addresses in the constrained area of memory is based at least

20   in part on a subset of bits of a corresponding one of the addressing values, reading

21   back 216 from the sequence of addressed locations values resulting from the storing

22   of the first sequence to obtain a second sequence of values, comparing 218 the first

23   sequence of values to the second sequence of values, selectively combining 220,

24   with an arithmetic-logical operation, certain elements of the second vector of

25   operand values based on results of the comparing, using at least some of the first

26   vector register of addressing values, loading 224 a third vector register with elements

27   from memory, performing 226 the arithmetic-logical operation using values from the

28   third vector register and the combined second vector of operand values to generate a

29   result vector, and using the at least some of the first vector register of addressing

30   values, storing 228 the result vector to memory.

1   In some embodiments, addresses of the elements from memory are calculated

2   by adding each respective addressing value to a base address.

3   In some embodiments, addresses of the elements from memory are calculated

4   by performing a signed-addition operation of each respective addressing value to a

5   base address of an object in memory.

6   In some embodiments, the arithmetic-logical operation is an addition

7   operation that produces at least one element of the result vector as a summation of an

8   element of the loaded third vector register and a plurality of respective elements of

9   the original second vector of operand values corresponding to elements of the first

10 vector register of addressing values having identical values.

11   In some embodiments, address values for the sequence of addressed locations

12 within the constrained area of memory are each calculated using a truncated portion

13 of each respective addressing value of the first vector register of addressing values.

14   In some embodiments, data values of the first sequence of values are each

15 formed by concatenating a portion of each respective addressing value of the first

16 vector register of addressing values to a respective one of a sequence of numbers.

17   In some embodiments, the constrained area contains 2^N consecutive

18 addresses, wherein address values for the sequence of addressed locations within the

19 constrained area of memory are each calculated using an N-bit value derived from

20 each respective addressing value of the first vector register of addressing values, and

21 wherein data values of the first sequence of values are each formed by concatenating

22 a portion of each respective addressing value of the first vector register of addressing

23 values to a respective one of a consecutive sequence of integer numbers.

24   In some embodiments, for the loading of the third vector register with

25 elements from memory, elements are loaded from locations specified by addressing

26 values corresponding to indications that indicated compares and no elements are

27 loaded from locations specified by addressing values corresponding to indications

28 that indicated miscompares.

29   Another aspect of the invention provides a computer-readable medium

30 having instructions stored thereon for causing a suitably programmed information-

15

1 processing system to execute one or more of the various embodiments of the above

2 method.

3   In some embodiments, the constrained area contains 2^N consecutive

4 addresses, address values for the sequence of addressed locations within the

5 constrained area of memory are each calculated using an N-bit value derived from

6 each respective addressing value of the first vector register of addressing values, data

7 values of the first sequence of values are each formed by combining at least a portion

8 of each respective addressing value of the first vector register of addressing values to

9 a respective one of a consecutive sequence of integer numbers, for the loading of the

10 third vector register with elements from memory, elements are loaded from locations

11 specified by addressing values corresponding to indications that indicated compares

12 and no elements are loaded from locations specified by addressing values

13 corresponding to indications that indicated miscompares, addresses of the elements

14 from memory are calculated by adding each respective addressing value to a base

15 address, the arithmetic-logical operation is a floating-point addition operation that

16 produces at least one element of the result vector as an ordered-operation floating

17 point summation of an element of the loaded third vector register and a plurality of

18 respective elements of the original second vector of operand values corresponding to

19 elements of the first vector register of addressing values having identical values, and

20 for the storing of the result vector of elements to memory, elements are stored to

21 locations specified by addressing values corresponding to indications that indicated

22 compares and no elements are stored to locations specified by addressing values

23 corresponding to indications that indicated miscompares.

24   Another aspect of the invention provides a system that includes a first vector

25 processor having a first vector register having addressing values, a second vector

26 register having operand values, a third vector register, a bit vector register, circuitry

27 that selectively stores a first sequence of values to a sequence of addressed locations

28 within a constrained area of memory, wherein each location's address is based at

29 least in part on a corresponding one of the addressing values, circuitry that

30 selectively loads, from the sequence of addressed locations, values resulting from the

16

1   stores of the first sequence to obtain a second sequence of values, circuitry that

2   selectively compares the first sequence of values to the second sequence of values to

3   generate bit values into the bit vector register representing compares and

4   miscompares, circuitry that selectively compresses the second vector of operand

5   values using the values in the bit vector register, circuitry that selectively loads the

6   third vector register with elements from memory addresses generated from the first

7   vector register of addressing values as masked by the bit vector register, circuitry

8   that selectively performs an arithmetic-logical operation on corresponding values

9   from the third vector register and the compressed second vector of operand values to

10   generate values of a result vector, and, circuitry that selectively stores the result

11   vector to memory.

12   Some embodiments of this system further include circuitry to calculate

13   addresses of the elements in memory by adding each respective addressing value to a

14   base address value.

15   In some embodiments of this system, the arithmetic-logical operation is an

16   addition operation that produces at least one element of the result vector as a

17   summation of an element of the loaded third vector register and a plurality of

18   respective elements of the original second vector of operand values corresponding to

19   elements of the first vector register of addressing values that had identical values.

20   Some embodiments of this system further include circuitry to calculate

21   address values for the sequence of addressed locations within the constrained area of

22   memory using a truncated portion of each respective addressing value of the first

23   vector register of addressing values.

24   Some embodiments of this system further include circuitry to generate data

25   values of the first sequence of values by joining a portion of each respective

26   addressing value of the first vector register of addressing values to a respective one

27   of a sequence of numbers.

28   Some embodiments of this system further include circuitry to generate

29   address values of the sequence of addressed locations within the constrained area of

30   memory by adding a base address to an N-bit portion of each respective addressing

1 value of the first vector register of addressing values, and circuitry to generate data

2 values of the first sequence of values by combining a portion of each respective

3 addressing value of the first vector register of addressing values with a respective

4 one of a consecutive sequence of integer numbers.

5   In some embodiments, the circuitry that selectively loads the third vector

6 register with elements from memory only loads element from locations specified by

7 addressing values corresponding to bits of the bit vector that indicated a compare.

8   Some embodiments further include synchronization circuitry that ensures that

9 the comparing the first sequence of values to the second sequence of values to

10 generate the bit vector representing compares and miscompares effectively

11 completes before the loading of the third vector register with elements from memory,

12 and that ensures that the storing the result vector to memory completes before

13 subsequent passes through a loop.

14   Some embodiments further include a second vector processor having: a first

15 vector register having addressing values, a second vector register having operand

16 values, a third vector register, a bit vector register, circuitry that selectively stores a

17 first sequence of values to a sequence of addressed locations within a constrained

18 area of memory, wherein each location's address is based at least in part on a

19 corresponding one of the addressing values, circuitry that selectively loads, from the

20 sequence of addressed locations, values resulting from the stores of the first sequence

21 to obtain a second sequence of values, circuitry that selectively compares the first

22 sequence of values to the second sequence of values to generate bit values into the

23 bit vector register representing compares and miscompares, circuitry that selectively

24 compresses the second vector of operand values using the values in the bit vector

25 register, circuitry that selectively loads the third vector register with elements from

26 memory addresses generated from the first vector register of addressing values as

27 masked by the bit vector register, circuitry that selectively performs an arithmetic-

28 logical operation on corresponding values from the third vector register and the

29 compressed second vector of operand values to generate values of a result vector,

30 and, circuitry that selectively stores the result vector to memory. This system also

1   includes synchronization circuitry that ensures that the comparing the first sequence

2   of values to the second sequence of values to generate the bit vector representing

3   compares and miscompares effectively completes in both the first and second vector

4   processors before the loading of the third vector register with elements from memory

5   in either processor, and that ensures that the storing the result vector to memory

6   completes before subsequent passes through a loop.

7   Another aspect of the invention provides a system that includes a first vector

8   register, a second vector register, a third vector register, a bit vector register, means

9   for loading the first vector register with addressing values, means as described herein

10   for loading the second vector register with operand values, means for storing a first

11   sequence of values to a sequence of addressed locations within a constrained area of

12   memory, wherein each one of these location's addresses in the constrained area of

13   memory is based at least in part on a subset of bits of a corresponding one of the

14   addressing values, means for loading from the sequence of addressed locations

15   values resulting from the storing of the first sequence to obtain a second sequence of

16   values, means for comparing the first sequence of values to the second sequence of

17   values, means for selectively combining, with an arithmetic-logical operation, certain

18   elements of the second vector of operand values based on results of the comparing,

19   means for loading a third vector register with elements from memory address

20   locations generated using at least some of the first vector register of addressing

21   values, means for performing the arithmetic-logical operation using values from the

22   third vector register and the combined second vector of operand values to generate a

23   result vector, and means for storing the result vector to memory.

24   Another aspect of the invention provides a system including a first vector

25   register that can be loaded with addressing values, a second vector register that can

26   be loaded with operand values, a third vector register that can be loaded with

27   operand values from memory locations indirectly addressed using the addressing

28   values from the first vector register, a circuit that determines element addresses of

29   the first vector register that have a value that duplicates a value in another element

30   address, a circuit that selectively adds certain elements of the second vector of

19

1   operand values based on the element addresses the duplicated values, a circuit that

2   uses indirect addressing to selectively load the third vector register with elements

3   from memory, a circuit that selectively adds values from the third vector register and

4   the second vector of operand values to generate a result vector, and a circuit that

5   selectively stores the result vector to memory using indirect addressing.

6   Some embodiments of this system further include an adder that generates

7   addresses of the elements from memory by adding each respective addressing value

8   to a base address.

9   Some embodiments of this system further include an adder that generates

10   addresses of the elements from memory by a signed-addition operation of each

11   respective addressing value to a base address of an object in memory.

12   In some embodiments, the circuit that selectively adds certain elements

13   performs one or more addition operations using those values from a plurality of

14   respective elements of the original second vector of operand values corresponding to

15   elements of the first vector register of addressing values having identical values.

16

17   **Multistreaming Aspects of Indirect Addressed Vector Add**

18   Another aspect of the invention provides a computerized method that

19   includes loading a first vector register with addressing values, loading a second

20   vector register with operand values, determining which, if any, element addresses of

21   the first vector register have a value that duplicates a value in another element

22   address, selectively adding certain elements of the second vector of operand values

23   based on the element addresses the duplicated values, loading, using indirect

24   addressing from the first vector register, elements from memory into a third vector

25   register, adding values from the third vector register and the second vector of

26   operand values to generate a result vector, and storing the result vector to memory

27   using indirect addressing.

28   In some embodiments, the set of operations (a), (b), (c), and (d) is performed

29   substantially in parallel in the plurality of processors, and the set of operations (e),

30   (f), and (g) is performed serially, one processor at a time.

1   Some embodiments further include executing an ordered Msync operation

2   before the set of operations (e), (f), and (g), and executing an end ordered Msync

3   operation after the set of operations (e), (f), and (g).

4   In some embodiments, the set of operations (a), (b), (c), and (d) is performed

5   substantially in parallel in the plurality of processors.

6   Some embodiments of the method further include:

7   executing a first barrier synchronization operation before the set of

8   operations (e), (f), and (g) in all of the plurality of processors,

9   executing a second barrier synchronization operation before the set of

10   operations (e), (f), and (g) in the second processor,

11   executing the set of operations (e), (f), and (g) in the first processor

12   and then executing a second barrier synchronization operation in the first

13   processor to satisfy the second barrier synchronization in the second

14   processor, and executing a third barrier synchronization in the first

15   processor, and

16   executing the set of operations (e), (f), and (g) in the second processor

17   and then executing a third barrier synchronization operation in the

18   second processor to satisfy the third barrier synchronization in the first

19   processor.

20   In some embodiments, the set of operations (a), (b), (c), and (d) is performed

21   substantially in parallel in the plurality of processors.

22   In some embodiments, the determining of duplicates includes:

23   generating each respective address value for a sequence of addressed

24   locations within a constrained area of memory containing 2^N consecutive

25   addresses using an N-bit value derived from each respective addressing

26   value of the first vector register,

27   generating each respective data value of a first sequence of values by

28   combining at least a portion of each respective addressing value of the

29   first vector register to a respective one of a sequence of integer numbers,

30   storing the first sequence of values to the constrained memory area

1 using the generated sequence of respective address values,

2   loading a second first sequence of values from the constrained

3 memory area using the generated sequence of respective address values,

4 and

5   comparing the first sequence of values to the second sequence of

6 values, and

7 wherein the loading of the third vector register includes loading elements

8 from locations specified by addressing values corresponding to indications of

9 positive compares from the comparing,

10 wherein addresses of the elements from memory are calculated by adding

11 each respective addressing value to a base address,

12 wherein the adding includes a floating-point addition operation that produces

13 at least one element of the result vector as an ordered-operation floating point

14 summation of an element of the loaded third vector register and a plurality of

15 respective elements of the original second vector of operand values corresponding to

16 elements of the first vector of addressing values having identical values, and

17 wherein for the storing of the result vector of elements to memory, elements

18 are stored to locations specified by addressing values corresponding to indications of

19 positive compares.

20

21 Another aspect of the invention provides a computerized method that

22 includes:

23 (a) within a first vector processor:

24   loading a first vector register in the first vector processor with

25 addressing values,

26   loading a second vector register in the first vector processor with

27 operand values,

28   determining which, if any, element addresses of the first vector

29 register in the first vector processor have a value that duplicates a value in

30 another element address,

1   selectively adding certain elements of the second vector of operand

2   values in the first vector processor based on the element addresses the

3   duplicated values,

4 (b) within a second vector processor:

5   loading a first vector register in the second vector processor with

6   addressing values,

7   loading a second vector register in the second vector processor with

8   operand values,

9   determining which, if any, element addresses of the first vector

10   register in the second vector processor have a value that duplicates a value

11   in another element address,

12   selectively operating on certain elements of the second vector of

13   operand values in the second vector processor based on the element

14   addresses the duplicated values,

15 (c) performing a synchronization operation that ensures that prior store

16 operations effectively complete in at least the second vector processor before the

17 following (d) operations,

18 (d) within the first vector processor:

19   loading, using indirect addressing from the first vector register,

20   elements from memory into a third vector register in the first vector

21   processor,

22   operating on values from the third vector register and the second

23   vector of operand values in the first vector processor to generate a first

24   result vector, and

25   storing the first result vector to memory using indirect addressing.

26 (e) performing a synchronization operation that ensures that the storing of the

27 first result vector effectively completes before the following (f) operations, and

28 (f) within the second vector processor:

29   loading, using indirect addressing from the first vector register,

30   elements from memory into a third vector register in the second vector

1   processor,

2   operating on values from the third vector register and the second

3   vector of operand values in the second vector processor to generate a

4   second result vector, and

5   storing the second result vector to memory using indirect addressing.

6

7   In some embodiments, each of the "operating on" functions includes adding.

8   In some embodiments, the adding includes a floating-point addition operation

9 that produces at least one element of the result vector as an ordered-operation

10 floating point summation of an element of the loaded third vector register and a

11 plurality of respective elements of the original second vector of operand values

12 corresponding to elements of the first vector of addressing values having identical

13 values.

14   In some embodiments, the determining of duplicates includes generating each

15 respective address value for a sequence of addressed locations within a constrained

16 area of memory containing 2^N consecutive addresses using an N-bit value derived

17 from each respective addressing value of the first vector register, generating each

18 respective data value of a first sequence of values by combining at least a portion of

19 each respective addressing value of the first vector register to a respective one of a

20 sequence of integer numbers, storing the first sequence of values to the constrained

21 memory area using the generated sequence of respective address values, loading a

22 second first sequence of values from the constrained memory area using the

23 generated sequence of respective address values, and comparing the first sequence of

24 values to the second sequence of values.

25   In some embodiments, the loading of the third vector register of each

26 processor includes loading elements from locations specified by addressing values

27 corresponding to indications of positive compares from the comparing operation.

28   In some embodiments, indirect addresses of the elements from memory are

29 calculated by adding each respective addressing value to a base address.

1　　　　One aspect of the invention provides a system that includes a first vector

2　　register having addressing values, a second vector register having operand values,

3　　circuitry programmed to determine which, if any, element addresses of the first

4　　vector register have a value that duplicates a value in another element address,

5　　circuitry programmed to selectively add certain elements of the second vector of

6　　operand values based on the element addresses the duplicated values, circuitry

7　　programmed to load, using indirect addressing from the first vector register, elements

8　　from memory into a third vector register, circuitry programmed to add values from

9　　the third vector register and the second vector of operand values to generate a result

10　　vector, and circuitry programmed to store the result vector to memory using indirect

11　　addressing.

12　　　　In some embodiments, the circuitry programmed to determine duplicates

13　　further includes circuitry programmed to generate each respective address value for a

14　　sequence of addressed locations within a constrained area of memory containing 2^N

15　　consecutive addresses using an N-bit value derived from each respective addressing

16　　value of the first vector register, circuitry programmed to generate each respective

17　　data value of a first sequence of values by combining at least a portion of each

18　　respective addressing value of the first vector register to a respective one of a

19　　sequence of integer numbers, circuitry programmed to store the first sequence of

20　　values to the constrained memory area using the generated sequence of respective

21　　address values, circuitry programmed to load a second sequence of values from the

22　　constrained memory area using the generated sequence of respective address values,

23　　and circuitry programmed to compare the first sequence of values to the second

24　　sequence of values; and the circuitry programmed to load the third vector register

25　　loads elements from locations specified by addressing values corresponding to

26　　indications of positive compares; addresses of the elements from memory are

27　　calculated by adding each respective addressing value to a base address; and the

28　　circuitry programmed to add includes a floating-point adder that produces at least

29　　one element of the result vector as an ordered-operation floating point summation of

30　　an element of the loaded third vector register and a plurality of respective elements

1   of the original second vector of operand values corresponding to elements of the first

2   vector of addressing values having identical values.

3   Some embodiments further include circuitry programmed to perform the set

4   of operations (a), (b), (c), and (d) substantially in parallel in the plurality of

5   processors, and circuitry programmed to perform the set of operations (e), (f), and

6   (g) serially, one processor at a time.

7   Some embodiments further include circuitry programmed to execute an

8   ordered Msync operation before the set of operations (e), (f), and (g); and circuitry

9   programmed to execute an end ordered Msync operation after the set of operations

10   (e), (f), and (g). Some such embodiments further include circuitry programmed to

11   perform the set of operations (a), (b), (c), and (d) substantially in parallel in the

12   plurality of processors.

13   Some embodiments further include circuitry programmed to execute a first

14   barrier synchronization operation before the set of operations (e), (f), and (g) in all of

15   the plurality of processors, circuitry programmed to execute a second barrier

16   synchronization operation before the set of operations (e), (f), and (g) in the second

17   processor, circuitry programmed to execute the set of operations (e), (f), and (g) in

18   the first processor and then executing a second barrier synchronization operation in

19   the first processor to satisfy the second barrier synchronization in the second

20   processor, and executing a third barrier synchronization in the first processor, and

21   circuitry programmed to execute the set of operations (e), (f), and (g) in the second

22   processor and then executing a third barrier synchronization operation in the second

23   processor to satisfy the third barrier synchronization in the first processor. Some

24   such embodiments further include circuitry programmed to perform the set of

25   operations (a), (b), (c), and (d) substantially in parallel in the plurality of processors.

26   Another aspect of the invention provides a system that includes

27   (a)   a first vector processor including means as described herein for loading a first

28   vector register in the first vector processor with addressing values, means for loading

29   a second vector register in the first vector processor with operand values, means for

30   determining which, if any, element addresses of the first vector register in the first

1   vector processor have a value that duplicates a value in another element address, and

2   means for selectively adding certain elements of the second vector of operand values

3   in the first vector processor based on the element addresses the duplicated values;

4   and

5   (b)   a second vector processor including means for loading a first vector register

6   in the second vector processor with addressing values, means for loading a second

7   vector register in the second vector processor with operand values, means for

8   determining which, if any, element addresses of the first vector register in the second

9   vector processor have a value that duplicates a value in another element address,

10   means for selectively operating on certain elements of the second vector of operand

11   values in the second vector processor based on the element addresses the duplicated

12   values,

13   (c)   means for performing a synchronization operation that ensures that prior

14   store operations effectively complete in at least the second vector processors before

15   the operations of the following (d) means,

16   (d)   within the first vector processor: means for loading, using indirect addressing

17   from the first vector register, elements from memory into a third vector register in

18   the first vector processor, means for operating on values from the third vector

19   register and the second vector of operand values in the first vector processor to

20   generate a first result vector, and means for storing the first result vector to memory

21   using indirect addressing;

22   (e)   performing a synchronization operation that ensures that the storing of the

23   first result vector effectively completes before the operations of the following (f)

24   means, and

25   (f)   within the second vector processor:

26   means for loading, using indirect addressing from the first vector

27   register, elements from memory into a third vector register in the second

28   vector processor,

29   means for operating on values from the third vector register and the

30   second vector of operand values in the second vector processor to generate

1   a second result vector, and

2   means for storing the second result vector to memory using indirect

3   addressing.

4   In some embodiments, each of the means for operating on functions includes

5   an adder.

6   In some embodiments, wherein the adder includes a floating-point adder that

7   produces at least one element of the result vector as an ordered-operation floating

8   point summation of an element of the loaded third vector register and a plurality of

9   respective elements of the original second vector of operand values corresponding to

10   elements of the first vector of addressing values having identical values.

11   In some embodiments, wherein the means for determining of duplicates

12   includes: means as described herein for generating each respective address value for

13   a sequence of addressed locations within a constrained area of memory containing 2^N

14   consecutive addresses using an N-bit value derived from each respective addressing

15   value of the first vector register, means for generating each respective data value of a

16   first sequence of values by combining at least a portion of each respective addressing

17   value of the first vector register to a respective one of a sequence of integer numbers,

18   means for storing the first sequence of values to the constrained memory area using

19   the generated sequence of respective address values, means for loading a second first

20   sequence of values from the constrained memory area using the generated sequence

21   of respective address values, and means for comparing the first sequence of values to

22   the second sequence of values.

23   In some embodiments, the means for loading of the third vector register of

24   each processor includes means for loading elements from locations specified by

25   addressing values corresponding to indications of positive compares from the

26   comparing operation.

27   In some embodiments, indirect addresses of the elements from memory are

28   calculated by adding each respective addressing value to a base address.

29   Another aspect of the invention provides a computer-readable medium

30   having instructions stored thereon for causing a suitably programmed information-

1   processing system to execute a method that includes loading a first vector register

2   with addressing values, loading a second vector register with operand values,

3   determining which, if any, element addresses of the first vector register have a value

4   that duplicates a value in another element address, selectively adding certain

5   elements of the second vector of operand values based on the element addresses the

6   duplicated values, loading, using indirect addressing from the first vector register,

7   elements from memory into a third vector register, adding values from the third

8   vector register and the second vector of operand values to generate a result vector,

9   and storing the result vector to memory using indirect addressing.

10   An iota instruction is described in U.S. Patent No. 6,308,250, entitled

11   "Method and Apparatus for Processing a Set of Data Values with Plural Processing

12   Units Mask Bits Generated by Other Processing Units," issued October 23, 2001 to

13   Klausler, the description of which is incorporated herein by reference.

14

1   In some embodiments, a program such as the following example is used:

2   ==

3

4   /* kernel of the HMG tabletoy benchmark (with declarations) */

5

6   #define LTABSIZE   22   /* logarithm of table size (27) (22

7   for jobmix) */

8   #define NRECGEN   100000   /* records to generate on each

9   pass */

10

11   #define TAB_SIZE   (1 << LTABSIZE)

12   double table[TAB_SIZE];

13

14   typedef struct

15   {

16   int index;

17   double value;

18   }update_t;

19

20   update_t xdata[NRECGEN];

21

22   ...

23

24   /* the timed loop, recs_todo (input data) = 900000000 */

25

26   while (recs_todo)

27   {

28   nrec = MIN(recs_todo, NRECGEN);

29   recs_todo -= nrec;

30

31   for (idx = 0; idx < nrec; idx++)

32   table[xdata[idx].index] += xdata[idx].value;

33   }

34

35   /* Please note that there is NO ivdep on this loop. */

36

```
1    /* In some embodiments, change the inner update loop to:

2

3     #pragma ivdep

4         for (idx = 0; idx < nrec; idx++)

5           table[xdata[idx].index]

6             += xpartred_add64(xdata[idx].value,xdata[idx].index);

7

8    /* in some embodiments, results were obtained by compiling with: */

9    /*    cc -o toy toy.c                                           */

10   /* and running with:                                           */

11   /*    aprun -n1 -p:16m toy 900000000                           */

12

13   ===============================================================

14

15   *   In some embodiments, the following assembly code is used for the

16   bolded instruction above:

17

18   *   HMG Tabletoy update:    table[xdata.index[i]] += xdata.value[i];

19

20   *   Registers computed or loaded during RHS processing of update...

21

22       v2      [a27,2],m0              ;IX = xdata.index[*]

23       v0      cidx(a11,m0)            ;IOTA

24    m1    m0|m0              ;input mask

25       v1      [a28,2],m0          ;Y = xdata.value[*]

26

27   *   Generate ordered msync wait,send masks

28   *

29   *   A10 = Remaining tripcount (after this pass)

30   *   A11 = 1

31   *   A22 = SSP#

32   *   A26 = SSP's array offset

33

34       a24     a22^3             ;=0 iff P3

35       a25     a0<a26            ;=0 iff P0 and 1st iter, else 1

36       a24     a10|a24           ;=0 iff P3 and last iteration

37       a21     a22-1
```

31

```
1        a26    a0<a24          ;=0 iff P3 and no more iters, else 1
2        a23    a22+1
3        a21    a21&3           ;restrict shift counts to be 0..3
4        a23    a23&3
5        a22    a11<<a22        ;self-mask
6        a21    a25<<a21        ;mask for SSP to wait on
7        a23    a26<<a23        ;mask for SSP to send
8        a21    a21|a22         ;wait mask
9        a22    a22|a23         ;send mask
10
11  *    Inlined "indexed partial reduction" algorithm:  Y',M1 =
12  reduce(Y, IX),M1
13  *
14  *    Y' will contain Y or sum reduced values of Y for duplicate IX
15  values;
16  *    M1 will contain an update mask where IX values are unique and
17  also where
18  *    the Y' elements that need to be added into the update (LHS)
19  vector.
20  *
21  *    Input:
22  *       v0 = IOTA vector (0,1,2,...,63)
23  *       v1 = Y vector
24  *       v2 = IX vector
25  *       m1 = Input mask
26  *       v1 = #elements in v0, v1, v2
27  *
28  *    Output:
29  *       v1 = Y' vector
30  *       v2 = IX vector
31  *       m1 = Output mask of unique IX values
32
33  CNFXSZ      =      16384     ;Size of scratch conflict analysis
34  space
35
36        s4     CNFXSZ-1
37        a29    v1
```

```
 1          a45    CNFXSZ*8-8
 2          v5     v2&s4,m0              ;Conflict index set masked from ix
 3          m4     fill(a29)
 4          m3     m1&m4                 ;Clear trailing mask bits beyond VL
 5          a20    CNFXSZ*8
 6          a45    a63-a45
 7          s28    8
 8          a63    a63-a20                   ;Allocate private stack space
 9
10          v6     v2<<s28,m0           ;(ix<<8) to make room for IOTA
11          v4     v6|v0,m0             ;(ix<<8)|IOTA
12          a27    last(m4)             ;last valid element#
13
14   cnfxloop =  *                      ;"False positive" conflict loop
15          [a45,v5] v4,m3,ord              ;Scatter (ix<<8)|IOTA (to
16   scratch array)
17          s27    x'00ff:d
18          lsync v,v
19          v6     [a45,v5],m3          ;Gather  (ix<<8)'|IOTA'
20
21          v7     +v6>>s28,m3          ;Extract ix'
22          m2     v7==v2,m3            ;M2 excludes ix's mapping to same CNFX
23
24          v9     v6&s27,m3            ;Element #s of y sums
25          m4     v9!=v0,m2            ;Conflict map
26          m3     ~m2&m3                   ;Map of remaining ix values
27
28          a6     1
29          a29    pop(m4)                  ;Conflict trip count (tc)
30
31          v7     cmprss(v9,m4)            ;IOTA's that conflicts map to
32          a26    pop(m3)                  ;>0 if ix's mapped to same CNFX
33          m1     ~m4&m1                   ;Exclude conflicts in final M1
34
35          a1     v7,0                ;1st iota into which to sum (iota1)
36          a8     a6<a29                   ;=1 if tc > 1
37          v7,a29      a27                 ;Store safe y sum index at end
```

33

```
1        a6      a0<a29                  ;=1 if tc > 0

2        a7      a6+a8           ;=2 if tc > 1, else tc

3

4        a2      v7,a6           ;2nd iota into which to sum (iota2)

5        a3      v7,a7           ;3rd iota into which to sum (iota3)

6

7        v8      cmprss(v1,m4)           ;y values to add into y sums

8        bz      a29,noconflict          ;If no conflicts exist

9

10       a11     v8,0            ;Get 1st 3 y values (y1,y2,y3)

11       v8,a29          s0              ;Store 0 for conflict summing at

12   end

13       a12     v8,a6

14       s3      v8,a7

15

16       $REPEAT                         ;Repeat 3 update fixes per

17   iteration

18           a5      a7<a29          ;=1 if >=0 more conflicts (another

19   iter)

20           s5      v1,a1                   ;Get 3 y sums (to sum conflicts

21   into)

22           a23     a2^a1                   ;Determine conflict:

23   iota2==iota1

24           a5      a7+a5

25           s6      v1,a2

26           a24     a3^a1                   ;Determine conflict:

27   iota3==iota1

28           a15     a5<a29          ;=1 if >=1 more conflicts

29           s7      v1,a3

30           a25     a3^a2                   ;Determine conflict:

31   iota3==iota2

32           a6      a5+a15

33

34           a16     a1              ;Save iota1

35           a1      v7,a5                   ;Bottom load next iter's iota1

36           a7      a6<a29          ;=1 if >=2 more conflicts

37           a17     a2              ;Save iota2
```

```
1          a2      v7,a6                      ;Bottom load next iter's iota2
2          a7      a6+a7
3          a18     a3                 ;Save iota3
4
5          a13     a11
6          s1      a11
7          a11     a24?a0:a11                 ;y1 if iota3==iota1, else 0
8          a3      v7,a7                      ;Bottom load next iter's iota3
9          a13     a23?a0:a13                 ;y1 if iota2==iota1, else 0
10         s2      a12
11         a12     a25?a0:a12                 ;y2 if iota3==iota2, else 0
12
13         s11     a11
14         a11     v8,a5                      ;Bottom load next iter's y1
15         s13     a13
16         s12     a12
17         a12     v8,a6                      ;Bottom load next iter's y2
18
19         s4,d            s3+s11             ;y3 += (iota3==iota1)? y1 : 0
20         s3      v8,a7                      ;Bottom load next iter's y3
21         s2,d            s2+s13             ;y2 += (iota2==iota1)? y1 : 0
22         s4,d            s4+s12             ;y3 += (iota3==iota2)? y2 : 0
23
24         s5,d            s5+s1                 ;Sum1 += y1
25         s6,d            s6+s2                 ;Sum2 += y2 [+ y1]
26         s7,d            s7+s4                 ;Sum3 += y3 [+ y1] [+ y2]
27         v1,a16  s5
28         v1,a17  s6
29         v1,a18  s7
30
31      $UNTIL       a15,Z
32
33 noconflict =        *                   ;Branch here if no conflicts
34      bn      a26,cnfxloop               ;Repeat if more ix's mapped to
35 same CNFX
36
37      a63     a63+a20                     ;Restore stack frame
```

```
*
*    End of inlined "indexed partial reduction" algorithm.
*
*    Update LHS using unique IX mask, M1, and non-allocating
gather/scatter.
*    Use ordered (ripple) msyncs if multistreamed.
*
        msync a21,v                ;Ordered msync
        v4    [a32,v2],m1,na           ;Gather TABLE[xdata.index[*]]
        v5,d  v4+v1,m1
        [a32,v2] v5,m1,ord,na           ;scatter my updated TABLE values
        msync a22,v                ;End ordered msync
```

It is understood that the above description is intended to be illustrative, and not restrictive. Many other embodiments will be apparent to those of skill in the art upon reviewing the above description. The scope of the invention should, therefore, be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled. In the appended claims, the terms "including" and "in which" are used as the plain-English equivalents of the respective terms "comprising" and "wherein," respectively. Moreover, the terms "first," "second," and "third," etc., are used merely as labels, and are not intended to impose numerical requirements on their objects.